L-SCN: Layered SCN architecture with supernodes and Bloom filters

Gasparyan, Mikael ; Braun, Torsten ; Schiller, Eryk

Abstract: In this paper, we present L-SCN, a new routing architecture for Service-Centric Networking (SCN), which makes use of a two-layer forwarding scheme composed of inter-domain and intra-domain communication. Unlike existing SCN routing architectures relying on a flat organization, our design splits the network into domains. Nodes within a domain possess significant knowledge about existing services and available resources within the domain. Supernodes provide a significant advantage in comparison to other architectures. They assure the inter-domain communication and make use of a pull and push mechanism combined with Bloom filters. It allows us to minimize the protocol overhead and optimize sharing of information about available services and resources in the network.

DOI: https://doi.org/10.1109/CCNC.2017.7983252

L-SCN: layered SCN architecture with Supernodes and Bloom Filters

Mikael Gasparyan
Institute of Computer Science
University of Bern
Bern, Switzerland
gasparyan@inf.unibe.ch

Torsten Braun
Institute of Computer Science
University of Bern
Bern, Switzerland
braun@inf.unibe.ch

Eryk Schiller
Institute of Computer Science
University of Bern
Bern, Switzerland
schiller@inf.unibe.ch

Abstract—In this paper, we present L-SCN, a new routing architecture for Service-Centric Networking (SCN), which makes use of a two-layer forwarding scheme composed of inter-domain and intra-domain communication. Unlike existing SCN routing architectures relying on a flat organization, our design splits the network into domains. Nodes within a domain possess significant knowledge about existing services and available resources within the domain. Supernodes provide a significant advantage in comparison to other architectures. They assure the inter-domain communication and make use of a pull and push mechanism combined with Bloom filters. It allows us to minimize the protocol overhead and optimize sharing of information about available services and resources in the network.

Keywords—service-centric networking; service routing; content-centric networking; bloom filter; supernode

I. INTRODUCTION

Currently, the Internet architecture is host-based. A requester has to possess the network address of the server to receive the data of interest. The content-centric networking paradigm aims to make content directly addressable. Its goal is to transform the current host-centric architecture to an information-centric one. The basic idea behind this concept is that a client sends interest requests for content without prior knowledge of the content location. Content-Centric Networking (CCN) [1] is one of the most influential information-centric architectures. There exist, however, many content-centric networking architectures derived from it, e.g., Named-Data Networking (NDN) [2].

Today's Internet is becoming more service oriented. Therefore, the future Internet has to provide integrated service support. CCN, as its name suggests, is centered on the idea of handling content requests. The goal of this paper is to design a Service-Centric Networking (SCN) [3] architecture derived from the CCN paradigm. The research on CCN continues for more than a decade. Therefore, there is a substantial number of CCN implementations. The research on SCN is new and there are many unsolved problems ahead of us. Some of the concepts, however, may directly inherit from CCN.

This paper has the following organization. Related work is handled in Section II. Section III describes the newly proposed architecture called L-SCN (Layered-SCN), demonstrates potential benefits, and presents an example of real applications. Section IV discusses the implementation and evaluation of L-SCN. Finally, we conclude and present future work in Section V.

II. RELATED WORK

SCN builds upon the CCN framework. There already exist some research works that cope with small sub-problems that emerge when CCN adapts to service centric scenarios (e.g., naming, service management) [4]. There are, however, only a few works that tackle the problem of SCN from a holistic architectural perspective. This section gives a short overview of currently available prominent examples of the service-centric networking architecture.

There are many works on service-oriented architectures based on a centralized paradigm, in which a centralized controller possesses a global view of the network to instantiate services and forward requests among appropriate instances. The centralized controller has up to date information about available services and current load. A single point of failure and heavy protocol overhead are, however, the main drawbacks of the centralized solutions [5, 6, 7].

CCNxServ [8] is built on top of CCNx [9], which is an implementation of the CCN architecture. CCNxServ provides NetServ [10], which is a software component extending CCNx with support for services. CCNxServ allows us to deploy services dynamically. A CCNxServ node has to fetch and deploy the service application from the network prior to execute the service request. The deployment of the service can become a main cause of performance issues. Moreover, CCNxServ does not fully comply with the CCN architecture because of NetServ, which is IP based.

Service over Content-Centric Routing (SoCCeR) [11] is one of the most prominent SCN architectures. SoCCeR extends CCN with integrated support for service routing decisions leveraging the Ant-Colony Optimization [12]. SoCCeR adds a control layer on top of CCN, which allows us to manage routing decisions. It modifies the CCN Data and Interest messages to enable a control layer, which is installed on all nodes and periodically broadcasts Ant-Interest messages for distinct services. A message can traverse the whole network until arriving at a node serving the corresponding service. The node handling the service replies to the Ant-Interest messages with an Ant-Data message, which contains server status information such as CPU usage or memory consumption. The

Ant-Data message will make use of breadcrumb information left by Ant-Interest messages to successfully arrive at the service requester. All the forwarding nodes will use the server status information gathered in Ant-Data messages. It will be stored in a specific data structure on the node to alter the content of the forwarding table.

Serval [13] is an architectural approach, which designs a new service access layer (SAL) that allows applications to communicate using service names. SAL is located between the transport and the network layer. Serval uses a service table to map service names onto network addresses for forwarding decisions. Moreover, Serval allows for load balancing and service maintenance. Special routers are responsible for finding the best available service to satisfy an incoming request. Due to the fact that Serval modifies the TCP/IP stack, the integration of Serval with the current Internet architecture is a difficult task.

NextServe [14] extends the CCN architecture with support for services. It allows service composition and uses a naming scheme to request for services. This idea resembles calling for object methods in the object-oriented programming paradigm. There is no implementation or evaluation of NextServe, only the description of the architecture is available.

Named Function Networking (NFN) [15] integrates a λ-expression resolution engine on top of CCN. Service requesters send an Interest message, which contains a data block and a set of functions that has to be executed on the data block. In [15], CCN was extended with NFN and a Service layer. The NFN layer is responsible for finding optimal resources or cached computations already present in the network. The Service layer recognizes the required resources and initiates computing.

III. L-SCN DESIGN

Our proposed solution is a two-layer architecture. In contrast to existing solutions, we provide significant differences in node clustering and server status information exchange. We cluster the network by introducing domains to reduce the protocol overhead dramatically and better share the information about available services or resources in the network. This is achieved through the integration of the push and pull mechanism combined with the introduction of supernodes and Bloom filters [16].

A. Introduction

L-SCN combines multiple forwarding mechanisms. The default CCN routing mechanism is not modified. Hence, traditional CCN traffic can be routed as usual.

In the designed architecture, nodes are clustered and placed within domains. The clustering process is based upon proximity, which does not necessarily mean geographical distance, but rather good connectivity by high capacity links.

The proposed communication is organized into inter-domain and intra-domain schemes. Nodes within the same domain can directly communicate with each other. In each domain, there is at least one supernode, which is responsible for the inter-domain communication and the aggregation of available service and resource information. Supernodes possess important information on their domains such as available resources and accessibility. The only mandatory requirement to become a supernode in a domain is to have a connection link with at least one supernode of another domain. Preferably, supernodes also have relatively high-capacity links to neighboring nodes. A push and pull communication mechanism using supernodes and Bloom filters assure inter-domain and intra-domain communication. Fig. 1 shows two domains with nodes (grey) and supernodes (black).

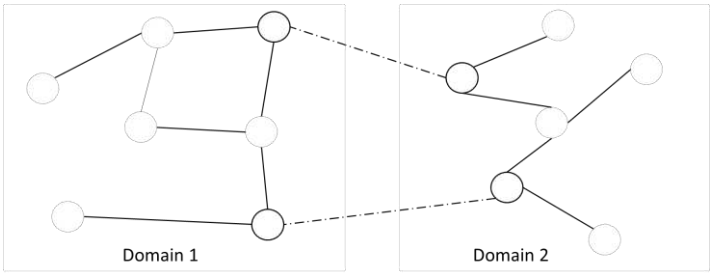

Fig. 1. Example of two connected domains

If a node from another domain requires detailed information on some services, it has to request it through the supernode. The supernode can directly answer to this request, because it possesses fresh information about all services available in its own domain. The following paragraphs explain our architecture, as well as the inter-domain and intra-domain communication mechanism in detail. We first present the communication protocol among nodes in the same domain and later on, the inter-domain communication mechanism.

B. Intra-Domain Communication

Supernodes periodically broadcast Interest Information messages (IIM) to all nodes of their domain. Service provider nodes reply with a Data Information Message (DIM). The response message contains an array with resource availability information and a Bloom filter storing all the services provided by the node. A Bloom filter is a probabilistic and space-efficient data structure conceived to achieve membership queries on a large dataset. It allows for tests whether an element is a member of the set with a certain error rate. False positive errors can occur, while false negative errors cannot occur. For example, a Bloom filter possessing a maximum false positive error rate of 1% and an optimal number of *k* hash functions storing 1000 elements requires only around 1 KB of storage space. Tables I and II show the content of the Interest and Data Information Messages.

TABLE I. INFORMATION STORED IN THE IIM

Interest Information Message (IIM)	
Name	The name of the IIM message starts with broadcast followed by the requester's unique node ID (e.g., \broadcast\S1\)
Timestamp	Timestamp of the IIM requester supernode

TABLE II. INFORMATION STORED IN THE DIM

Data Information Message (DIM)	
Name	The name of the DIM message starts with broadcast followed by the requester's unique node ID (e.g., \broadcast\S1\)
Timestamp	Timestamp of the DIM provider node, it is updated when sent from the cache of an intermediate node
Available Services	Bloom filter storing the available services
Available Resources	Available resources of the node (e.g., cpu, ram)
Node ID	The intra-domain node identifier (e.g., node4)

The DIM response follows the path of the received IIM message towards the requesting supernode. The information on available services is stored in Bloom filters together with the service status information. All intermediate nodes on the way update their local tables with the information gathered from the reply messages. This is used for service request forwarding decisions. The Interest and Data Information messages allow the supernodes to discover the intra-domain services, their availability (through what faces), and the server status information (e.g., cpu load, memory consumption). It allows us to forward the requests accordingly. To reduce the protocol overhead in the case when a node receives the same broadcast message multiple times, the broadcast requests are equipped with a unique identifier. The forwarding node does not forward the same IIM message over the same face again.

The periodic broadcast IIM messages are sent at the end of a specified time window. There is no explicit synchronization of time windows among different supernodes. However, the synchronization occurs implicitly, since a supernode never forwards incoming IIM messages from other supernodes. Instead, it either starts sending its own IIM message or directly replies with recently received and stored DIM messages. The following paragraph gives an example of how the implicit IIM broadcast synchronization happens in a cell of two supernodes A and B.

A specified time window (e.g., 10 seconds) is used by the supernodes to periodically broadcast IIM requests. When A starts sending its IIM requests, and the message arrives at B, B has the following obligations. If B has performed broadcasting before, it replies to A with its locally stored DIMs. Otherwise, it begins broadcasting its own IIM requests and puts the request from A in its Pending Interest Table (PIT). As defined in CCN, PIT keeps track of received but not yet answered Interest requests. Afterwards, when the DIM responses arrive, it will forward them to A.

In the following, we illustrate the previously presented ideas through an example depicted in Fig. 2. There are two supernodes S1-S2 in black and five regular nodes N1-N5 in grey. If the broadcast time window in S1 expires, S1 starts sending a regular broadcasting IIM request. In this example, the broadcast arrives at N2 and N5 and they reply to S1 with a DIM messages. N1 and N5 put the request of S1 into their PIT (\broadcast\S1\) and forward the request to all faces except the incoming face of the requesting IIM. Then, N1 and N3 receive the request from N2, and reply with their DIM.

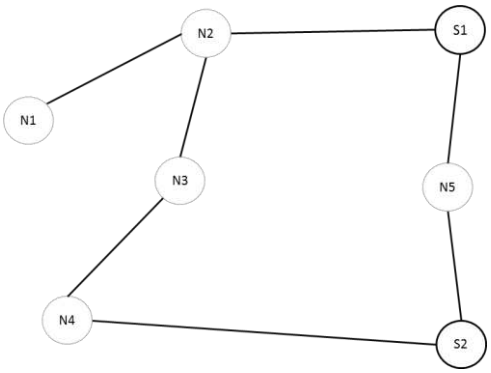

Fig. 2. Example domain topology with two supernodes and five ordinary nodes

N3 puts the request in its PIT, because it will forward it to N4. When N2 receives responses from N1 and N3, it caches and forwards them to the requester S1. The caching is of high importance, because a broadcast arriving from another node within a short time period can be directly answered with cached DIMs. Caching relieves us from an execution of a costly forwarding procedure again.

When S2 receives S1 IIM request through N5 the following can happen. If S2 has recently sent an IIM, it can reply with previously received DIMs. If IIMs were not yet issued, S2 starts its IIM sequence. The S1 IIM request goes to the S2 PIT to forward incoming DIMs to S1 until the IIM broadcast of S2 is completed.

At the end of the supernode broadcast process, the supernodes possess the DIMs of all the nodes in the domain and the faces required to reach a given node. The DIM contains an array, which stores the information on available resources and a Bloom filter with available service names.

Table III summarizes the information stored by the nodes in a domain. The supernodes have a complete view of the domain, while regular nodes have only partial information, i.e., an regular node only knows the server status information of the fraction of clients, whose DIMs traversed this node.

TABLE III. INFORMATION STORED IN THE NODES

Information stored by ordinary nodes and supernodes in the domain	
Service Provider's Node ID	Unique identifier of the service provider node (e.g., node4)
Face ID	Face to reach the node
Available Services Bloom Filter	A Bloom filter containing the services provided by the node
Available Resources	Available resource information

Using this information from the DIMs, an upcoming service request can be forwarded to an appropriate node. If S1 has to forward a service request for a given Service A (example topology in Fig. 2), it searches through available resource information and Bloom filters to find the best face (leading to the service provider with the most available resources). This face will not only be used for request forwarding, but will also be stored inside the standard Forwarding Information Base (FIB) table. As defined in CCN, FIB is a table storing the forwarding face set for different name prefixes. It allows us to

forward future service requests immediately without searching through Bloom filter data structures again.

We also employ other protocol optimization techniques including caching and broadcast message identification to dramatically reduce overhead. 1) If a recent service status is available on a node, the broadcast is not forwarded, while the node can directly answer the request by using its cache. 2) While broadcast messages from a supernode are unique, they will not be processed twice on the same node.

C. Inter-Domain Communication

We have previously explained a mechanism to disseminate information within the same domain, i.e., intra-domain communication. A similar information broadcasting process disseminating available services and resources to other domains could definitely cause a significant amount of traffic in the network. We, therefore, decided to only broadcast Bloom filters that inform about services available in the domains. The information about available resources is not directly provided, and has to be explicitly requested.

Nodes in the network are connected with each through their faces. An intra-domain face is a face connecting two nodes from the same domain. However, an inter-domain face connects two supernodes from two distinct domains. The inter-domain faces are used to send Bloom filters as broadcast messages that allow supernodes to fill out their forwarding tables, which are used to forward requests along the inter-domain faces that lead towards the requested services.

The received broadcast message will not always be forwarded. The intermediate supernode will discard the message if the received Bloom filter is a subset of the previously forwarded Bloom filters. Please notice that the forwarding process forwards a Bloom filter, which becomes a union of all the Bloom filters received so far from the neighboring domains.

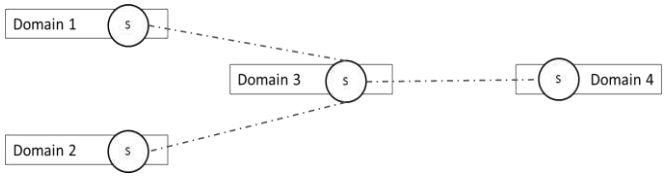

Fig. 3. Example of four domains linked with supernodes

Fig. 3 illustrates this process. Domain 3 receives a Bloom filter, which informs about available service names gathered by supernodes of Domains 1 and 2. Let us assume that Domain 1 sent its Bloom filter first. It was immediately forwarded by Domain 3 to Domain 4. Later on, when Domain 2 sent its Bloom filter to Domain 3, it had to go through a check, as it might be a subset of the already received and forwarded Bloom filter of Domain 1. If the Bloom filter of Domain 2 is only a subset of the information already provided by Domain 1, it has to be discarded. Otherwise, Domain 3 will prepare a union of Bloom filters issued by Domains 1 and 2 that will be forwarded to Domain 4.

To summarize, due to this broadcasting procedure, supernodes in the network get the information about inter-domain faces leading to services provided. The information about available resources on the nodes is, however, not provided at this stage. Moreover, the inter-domain broadcasting process is optimized using two conditions. 1) If the set of services provided did not change since the last forwarding, then supernodes do not have to forward Bloom filters containing the available services again. 2) As already mentioned, if an incoming broadcast is a subset of already broadcasted Bloom filters, it will not be forwarded further on.

To receive available resources for a given service, nodes send a Service Resource Information Interest (SRII). The Interest is forwarded through supernodes in the direction of a domain having a given service available. The destination supernode of the target domain will receive the request and reply to it with a Service Resource Information Data (SRID) message for the requested service. The response will follow back the path of the corresponding Interest message, while intermediate nodes will also cache this information for future forwarding decisions. SRID informs the nodes about forwarding faces leading to a node with the best available resources to handle a given service.

D. Service Requests

In this subsection, we present the intra-domain and inter-domain service requests. The main difference between them is that an intra-domain service request is equipped with additional information, which is the internal domain nodeID of the service provider.

An intra-domain service Interest request uses the following fields: the request prefix (e.g., “service”), nodeID (e.g., “node4”), the service identifier (e.g., “getWeather”), and a parametersID (described in the following paragraph e.g., “098f6bcd4621d373cade4e832627b4f6”) Therefore, the full intra-domain request can look like this: /service/node4/getWeather/098f6bcd4621d373cade4e832627b4f6). Additionally, the service request contains an input parameter data structure. The intermediate nodes know how to forward an incoming request, because they possess cached DIM replies collected upon the broadcast forwarding process. They do not need to search in stored Bloom filters, because the node can simply forward the request to an appropriate face using the nodeID. Roughly speaking, the aim of the nodeID in the intra-domain request allows intermediate nodes to save time by not using cached Bloom Filters for forwarding decisions. The intermediate nodes can directly forward the service request to an appropriate face by using the nodeID. Contrarily, an inter-domain service Interest request does not have a nodeID in its name.

A service Interest request also contains a parametersID, which is a unique identifier of the service request input parameters created with a hashing algorithm. A traditional CCN content request is identified by its name, but a request for a service has to also include input parameters. To identify the uniqueness of a request, we employ a service name and a hash (delivered through a hash function) of the associated input parameters. The hash is stored in a service request name

(parametersID). Currently, we employ MD5-sum hash function to populate the parametersID identifier. This allows us to identify the uniqueness of a service request and enables caching operations similar to regular content caching of CCN. An alternative service request naming scheme solution enabling caching could be implemented by storing all the service parameters inside the request name. This solution, however, does not allow us to request services with complex input parameters.

The regular CCN tables are not modified meaning that the content traffic can usually be forwarded. We, however, extended the regular CCN implementation with additional tables to store additional information required by our protocols. The table Extended Forwarding Information Base (EFIB) gathers information provided by DIM messages, i.e., the Bloom filters, the corresponding available resources, and the face through to access the responding node (i.e., DIM incoming face). This information is used to set an optimal forwarding face in the Forwarding Information Base (FIB) table, i.e., the information in EFIB is used to set the best forwarding face in the FIB table for future service forwarding. Afterwards, the arriving service requests (e.g. /service/ServiceName/parametersID) can be forwarded along the face leading to the server with the most free available resources. When the request reaches a supernode of the domain providing the service, it will be directly forwarded to the service provider node, as the supernode has full knowledge of services, available resources, and nodeIDs in its domain.

As an example of a service request, let us use a hypothetical getAverageWeatherByMonth service, which provides monthly average weather for a given geographical location. It accepts two parameters: month and GPS coordinates. An example inter-domain request would look like /service/getAverageWeatherByMonth/5a105e8b9d40e1329780d62ea2265d8a. Again, the hash in the request name is generated from the input parameters stored in the Interest packet in the same way as in the case of intra-domain requests.

IV. Evaluation and Results

In this section, we present a preliminary evaluation of our architecture compared to the three forwarding strategies integrated in NDN: Random, Multicast, and Best Route. As its name suggests, the random strategy randomly forwards incoming requests along one of the faces that lead to the service requested. The multicast strategy forwards the incoming request to all faces leading to the service requested, while the Best Route forwarding strategy forwards a request to the lowest cost forwarding node established by using network-based metric values [17].

A. Evaluation

We have implemented our architecture in ndnSIM [18] version 2.1. NdnSIM is a ns-3 based simulator for NDN. We evaluated the performance of our implementation in a simulator. However, even though ndnSIM does not use Direct Code Execution, our code requires only a small amount of modifications to run on a real system. We have modified the Interest and Data packets in ndnSIM, and new forwarding mechanisms were implemented. The existing information storage was extended with new data structures required.

We have evaluated the designed protocol implementation on a testing topology (Fig. 4) containing 100 nodes divided into 10 domains. This topology was selected, as it resembles the topology of the Internet at the small scale (with routers of high connectivity in the center and regular leaf-nodes with only one link provided).

Node clustering is out of scope of this document. It was, however, performed by a real world clustering algorithm, where ordinary nodes get connected to supernodes of the best connectivity. Every domain is equipped with one supernode. We have randomly selected 10 leaf nodes as service consumers and 15 leaf nodes as service providers. Service consumers send service requests using the random exponential function with the mean value equal to 1 second. Service providers in turn process the service requests coming from service requesters. The processing time of a service request is uniformly distributed between 1 and 2 seconds. Moreover, the resources of the processing node are consumed for that period. The simulations measure the service request processing time. We define the processing time as the time elapsed from the moment a request is sent by the consumer until the processed response is delivered back by the service provider.

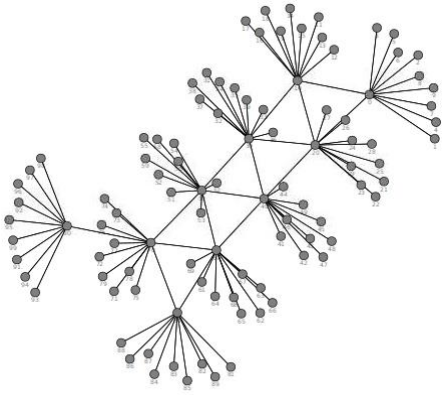

Fig. 4. The used topology in the evaluation with 100 nodes clustered into ten domains

In our scenario, each service consumer sends 100 unique service requests, which gives us a total number of 1000 unique service requests to be processed in the network per simulation round. We have repeated the simulation 10 times to establish the mean execution time and confidence intervals.

B. Results

We analyzed the processing time of L-SCN compared against the Random, Multicast and Best Route forwarding strategies existing in ndnSIM. Fig. 5 shows the mean processing time denoted by circles and confidence intervals labeled by horizontal lines. The confidence intervals for the different approaches do not overlap, which suggests significant difference between the mean execution time of the four aforementioned strategies. With high confidence the mean execution time of the executed scenario resides between the narrow confidence limits.

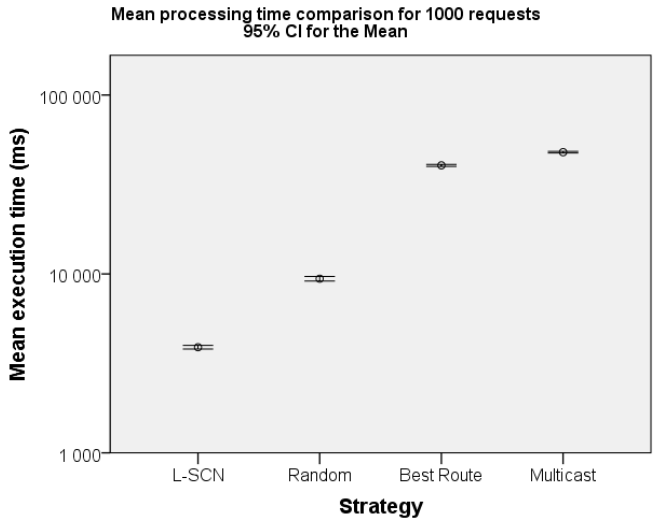

Fig. 5. Mean execution time for 1000 requests, comparing L-SCN to the forwarding strategies implemented in ndnSIM

In the evaluation, L-SCN has demonstrated a mean processing time for the 1000 requests of 3895ms (±94ms). It significantly outperforms the remaining three integrated forwarding strategies of ndnSIM.

The relatively high mean processing time of the multicast strategy is due to the fact that every service request will reach multiple nodes and therefore will be processed multiple times in the network. The Best Route strategy of ndnSIM selects the Best Face for forwarding based on network-based metric values (e.g., hop count, delay). It was developed mainly for content forwarding; Fig. 5 proves that it is not possible to achieve load balancing for Interest requiring processing in this scheme. The Random strategy of ndnSIM was the most efficient providing a mean processing time of 9404ms (±280ms). Nevertheless, L-SCN outperforms it by a factor of 2.4 with a mean processing time of 3895ms (±94ms). The evaluation of our implementation in ndnSIM shows that L-SCN functions as expected and produces outstanding results when compared to other prominent contributions in this domain.

V. Conclusions and Future work

To the best of our knowledge, the L-SCN is the first SCN architecture combining a two layer design, supernodes, and Bloom filters to optimize service accessibility, load balancing, and protocol overhead. We do not require a global coordinator for service requests to reach their respective providers. The nodes share the information on the services provided through Bloom filters and push and pull mechanisms. This allows for a proper balance between protocol overhead and amount of exchanged information.

We have implemented and evaluated our architecture in ndnSIM by comparing its performance with the Random, Best Route, and Multicast forwarding strategy available in ndnSIM. The simulation results show that L-SCN outperforms the three forwarding strategies implemented in ndnSIM.

We are planning to extend L-SCN with a significant amount of SCN features such as session support. It is worth noting that L-SCN is composed of domains that could further simplify the integration of sessions, while supernodes can act as session coordinators. We are also going to evaluate the scheme efficiency by varying the number of supernodes in the network, study the varying cluster size, and scalability by changing the network size. Another challenging problem to be considered is the strategy used for supernode selection.

References

[1] V. Jacobson, “A New Way to Look at Networking”, Google Tech Talk, August 2006.

[2] "Named Data Networking (NDN) - A Future Internet Architecture", Named Data Networking (NDN), 2016. [Online]. Available: http://named-data.net.

[3] T. Braun, V. Hilt, M. Hofmann, I. Rimac, M. Steiner and M. Varvello, "Service-Centric Networking", 2011 IEEE International Conference on Communications Workshops (ICC), 2011.

[4] T. Braun, A. Mauthe and V. Siris, "Service-centric networking extensions", Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC '13, 2013.

[5] R. Dutta, G. N. Rouskas, I. Baldine, A. Bragg and D. Stevenson, "The SILO Architecture for Services Integration, controL, and Optimization for the Future Internet," 2007 IEEE International Conference on Communications, Glasgow, 2007, pp. 1899-1904.

[6] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System", 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009.

[7] "AWS | Amazon EC2 – Service d'hébergement cloud évolutif", Amazon Web Services, Inc., 2016. [Online]. Available: https://aws.amazon.com/fr/ec2/

[8] S. Srinivasan, A. Singh, D. Batni, J. Lee, H. Schulzrinne, V. Hilt and G. Kunzmann, "CCNxServ: Dynamic service scalability in information-centric networks", 2012 IEEE International Conference on Communications (ICC), 2012.

[9] CCNx | PARC's implementation of content-centric networking", Ccnx.org, 2016. [Online]. Available: http://www.ccnx.org.

[10] S. Srinivasan, J. Lee, E. Liu, M. Kester, H. Schulzrinne, V. Hilt, S. Seetharaman and A. Khan, "NetServ", Proceedings of the 2009 workshop on Re-architecting the internet - ReArch '09, 2009.

[11] S. Shanbhag et al. "SoCCeR: Services over content-centric routing." Proceedings of the ACM SIGCOMM workshop on Information-centric networking. ACM, 2011.

[12] M. Dorigo and T. Stützle, Ant colony optimization. Cambridge, Mass.: MIT Press, 2004.

[13] Nordström, E., Shue, D., Gopalan, P., Kiefer, R., Arye, M., Ko, S. Y., ... & Freedman, M. J. (2012, April). Serval: An end-host stack for service-centric networking. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (pp. 7-7). USENIX Association.

[14] D. Mansour, T. Braun and C. Anastasiades, "NextServe Framework: Supporting Services over Content-Centric Networking", Lecture Notes in Computer Science, pp. 189-199, 2014.

[15] C. Tschudin and M. Sifalakis, "Named Functions for Media Delivery Orchestration", 2013 20th International Packet Video Workshop, 2013.

[16] B. Bloom, "Space/time trade-offs in hash coding with allowable errors", Communications of the ACM, vol. 13, no. 7, pp. 422-426, 1970.

[17] "Forwarding Strategies," - NdnSIM Documentation, 2016. [Online]. Available: http://named-data.net//2.1/fw.html.

[18] S. Mastorakis, A. Afanasyev, I. Moiseenko and L. Zhang, "ndnSIM 2.0: A new version of the NDN simulator for NS-3", NDN, Technical ReportNDN-0028, 2015